RECEIVED
CENTRAL FAX CENTER
NOV 02 2005

ANDERSON & JANSSON, LLP

9501 N Capital of Texas Hwy #202
Austin, TX 78759
512 372 8440 · 678 868 0101 (fax)
pehr@anjanlaw.com

Margaret Anderson
Pehr Jansson

FACSIMILE COVER SHEET

To: US Patent & Trademark Off. **FAX: 571 273 8300** **Attn: Commissioner for Patents** **MAIL STOP:** **Appeal Brief-Patents** **ART UNIT : 2192** **Examiner: RUTTEN, James D.** **From: Pehr Jansson** **Reg. No. 35,759**	**Certificate of Transmission under 37 CFR 1.8** I hereby certify that this correspondence is being facsimile transmitted to the United States Patent and Trademark Office (Fax No.: 571 273 8300) on November 2, 2005. Pehr Jansson
In regard to: Appl. No. : 09/992,558 Conf. No. : 5240 Applicant : Krishna Filing Date : 11/14/2001 Docket No. : 40.0042 Customer No. : 41754	This certificate applies to the following documents transmitted herewith: • Certificate of Transmission/Cover Sheet (this page) • Appeal Brief (41 pages)
Total number of pages including this cover page	-- 42 --

Application No. 09/992,558
Appeal Brief Filed on November 2, 2005

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application No.	:	**09/992,558**
Applicant	:	**KRISHNA, Ksheerabdhi**
Filing Date	:	**11/14/2001**
Confirmation No	:	**5240**
Art Unit	:	**2192**
Examiner	:	**RUTTEN, James D.**
Docket No.	:	**40.0042**
Customer No.	:	**41754**

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATE OF TRANSMISSION UNDER 37 CFR 1.8

Date of Transmission: November 2, 2005

I hereby certify that this correspondence is being facsimile transmitted to the United States Patent and Trademark Office at fax no. 703/872-9306 addressed to Mail Stop Appeal Brief, Commissioner for Patents, on the date indicated above.

Pehr B. Jansson

APPELLANT'S BRIEF

1. Real Party in Interest

The real party in interest in this appeal is Axalto, S.A., a corporation of France. The application is formally assigned to Schlumberger Malco, Inc. However, Schlumberger Limited, the former parent company of Schlumberger Malco, Inc. has divested itself of that part of its business. In that divestiture Axalto was formed as an independent company owning this intellectual property. A new assignment document to reflect this change was filed on August 10, 2005, with the assignment division of the U.S. Patent and Trademark office.

2. Related Appeals and Interferences

There are no related appeals and interferences.

3. Status of Claims

Claims 1-79 are pending in the application. Claims 1-79 were rejected in the office action of 05/02/2005 under 35 USC 103(a). Applicants appeal the rejection.

4. Status of Amendments

An amendment, amending the drawings, was filed on September 2, 2005. That amendment addressed a drawings objection in the final office action of 05/02/2005. In the Advisory Action of 9/29/2005, the Examiner entered the amendments to the drawings. Appellants infer that the drawings are now acceptable.

5. Summary of Invention

Appellants have invented a novel and non-obvious method for improving the loading of applets onto a smart card for execution. "The process of loading an applet onto a smart card for execution requires that the source code of the applet first be converted in to a corresponding binary representation of the classes making up the applet" (Specification, page 2, lines 8-10). "A CAP file consists of a set of components ... one such component is the Method Component which defines each of the methods declared in the package that makes up the CAP file" (Specification, page 2, lines 15-19). A method component contains (1) size item identifying the number of bytes of the method component, (2) a handler count indicating the number of exception handler entries for the method component, (3) an exception handler item, and (4) a method item which defines each variable length method body. Specification, Page 2, lines 16-29. "The method header does not specify the size of the associated method body. "The bytecodes within each method body typically contain operands consisting of various symbolic or unresolved code references which must be resolved prior to execution" (Specification, page 2, line 34 – page 3, line 1).

In the prior art, information allowing the resolution of disparate code references relied on providing a list of byte code offsets in a Reference Location Component and/or a Descriptor Component. Specification, Page 3, lines 11-30. "The Descriptor Component contains information on the boundaries of each method body so as to permit parsing and examination of the bytecodes contained within each method body" (Specification, page 3, lines 26-29.) The invention provides a novel and non-obvious mechanism for allowing "the contiguous bytecodes making up the methods item of the Method Component to be examined (and if necessary, their operands resolved) without reliance upon the Descriptor Component" (Specification, page 3, line 32-page 4, line 2). However, because the method header does not specify the size of the associated method body and if the Descriptor Component is not available, it is desirable to determine in some other way the extent of each method body. That is the invention being claimed.

Consider the following simple code segment for purposes of illustration:

```
Chapter 0 Method Header I
instr 1
instr 2
if variable a, goto instr 7
instr 4
instr 5
RETURN
instr 7
instr 8
RETURN
Method Header II
--
```

TABLE 1: Sample Code I

In the example of Table 2, the code from lines 1 through 9 represent one method body that may need to be examined for operands to be resolved. Without a Descriptor Component, it is not possible to know the end of the method body, thereby knowing when to end the examination. This is further complicated with exception handlers. Thus, it is desirable to have a way to determine that instruction 9 is the "Farthest Logical Return" from the method. Human visual inspection of the code sample will make it apparent to most skilled artisans that the farthest logical return is instruction 9. However, by sequential examination, that is not evident.

Claim 1 recites a solution for "A method for determining instruction boundaries of at least one method body within a computer code loaded into a memory of a smart card." The method comprises "examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction." While examining "maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected" and "terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer." Thus, determining instruction boundaries is accomplished by maintaining an FLR corresponding to the farthest forward jump instruction or the farthest valid ending instruction and terminating the examination when the next instruction to be examined is beyond the FLR.

Let's consider this method in the context of the Sample Code of Table 1. When the sequential examination hits the instruction at line 3, the FLR is set to 7,

which is at that point the farthest forward jump. The examination step proceeds with instructions 4, 5, and 6, until the examination process arrives at instruction 7. At that point, the Farthest Logical Return is still instruction 7. However, because it is not a logical return instruction, the Farthest Logical Return would be the next instruction, i.e., instruction 8. The process would then continue until instruction 9. Instruction 9 is a RETURN instruction. Thus, for it, the next instruction could not be a logical return instruction. Therefore, the FLR pointer would inherently not be advanced at that point. Consequently, when the process advances to Line 10, the "instruction under examination is beyond the instruction corresponding to the FLR Pointer.

6. Grounds for Rejection to be Reviewed on Appeal

35 USC 103(a)

Claims 1-3, 16-18, 44-47, and 62-65 were rejected under 35 USC 103(a) as unpatentable over Lance in view of Lai (Applicants adopt the Examiners short-hand form for all references discussed herein). Claims 4-6, 19-21, 48-50, and 66-68 were rejected under 35 USC 103(a) as unpatentable over Lance and Lai in view of Baentsch. Claims 7-9, 15, 22-24, 51-54, 61, and 69-72 were rejected as unpatentable over Lance and Lai in view of JCVMS 2.1. Claim 14 was rejected as unpatentable over Lance and Lai in view of JCVMS 2.1.1. Claims 10-12, 25-27, 55-57, and 73-75 were rejected as unpatentable over Lance, Lai and JCVMS in view of Baentsch. Claims 13, 28, 58, and 76 were rejected under 35 USC 103(a) as unpatentable over Lance, Lai, JCVMS 2.1 in view of JCVMS 2.1.1 and Lindholm. Claims 29-31, 59, and 77 were rejected under 35 USC 103(a) as unpatentable over Lance, Lai, in view of Chen. Claims 32-34 were rejected under 35 USC 103(a) as unpatentable over Lance, Lai, Chen in view of Baentsch. Claims 38-40 were rejected under 35 USC

103(a) as unpatentable over Lance, Lai, Chen and JCVMS 2.1 in view of Baentsch. Claim 41 was rejected under 35 USC 103(a) as unpatentable over Lance, Lai, Chen and JCVMS 2.1 in view of Lindholm. And finally, Claims 42, 60, 78 were rejected under 35 USC 103(a) as unpatentable over Lance, Lai, Chen in view of JCVMS 2.1.1.

The following table summarizes these rejections (a check ✓ signifies that the reference was used in the combination of references cited against the group of claims):

Claims	Lance	Lai	Baentsch	JCVMS 2.1	JCVMS 2.1.1	Lindholm	Chen
1-3, 16-18, 44-47, 62-65	✓	✓					
4-6, 19-21, 48-50, 66-68	✓	✓	✓				
7-9, 15, 22-24, 51-54, 61, 69-72	✓	✓		✓			
14	✓	✓			✓		
10-12, 25-27, 55-57, 73-75	✓	✓	✓	✓			
13, 28, 58, 76	✓	✓		✓	✓	✓	
29-31 59 77	✓	✓					✓
32-34	✓	✓	✓				
35-37, 43, 79	✓	✓		✓			
38-40	✓	✓	✓	✓			
41	✓	✓		✓		✓	✓
42, 60, 78	✓	✓			✓		✓

Table 2: Summary of Rejections under 35 USC 103(a)

7. Argument

35 USC 103

Claims 1, 16, 29, 59, 62, 77

This group of claims stands rejected under 35 USC 103(a) as unpatentable over Lance in view of Lai. Appellants respectfully request the reversal of the rejection of this group of Claims.

Claim 1 is representative of this group of claims as the other claims in the group contain limitations analogous to those recited in Claim 1.

The Examiner has failed to establish a prima facie case of obviousness. "To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations." MPEP 2143. The Examiner has failed to meet this burden.

Appellants respectfully request the Board to consider each of these criteria in turn: "First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings." Lance teaches "a new approach to building front-ends for automated program analysis tools." Lance, Page 1, Col. 1, lines 8-9. Lance teaches a method in which "the bytecode output of a standard, and unmodified, Java compiler can be analyzed to derive the basic data required to produce end-product program analysis information." Lance, Page 1, Col. 2, Lines 28-30. Lance uses a partitioning algorithm to break the code into basic blocks ("Our partitioning algorithm is an extension of the algorithm given by Aho, Sethi, and Ullman for basic block generation from three-address intermediate compiler code"). Lance, Page 2, Col. 1, liens 9-12.

It could be useful to also consider what Lance does in the context of the example of Table 1 above. Lance breaks a piece of code into basic blocks. Their "algorithm begins with identifying instructions that are *leaders*." Lance, page 3, Col. 1., lines 13-14. "An instruction is considered a leader if the instruction is the first instruction, the instruction is the target of a conditional or unconditional branch

instruction, or the instruction follows a conditional or unconditional branch instruction." Lance, page 3, Col. 1, lines 14-18. Thus, in the example of Table 1, instruction 1 (the first instruction), instruction 4 (the instruction following a branch), and instruction 7 (the target of a branch) are leaders. Next, the "algorithm partitions the code into basic blocks. A basic block is defined as consisting of the leader instruction and all instructions up to but not including the next leader instruction." Lance, page 3, Col. 1, lines 19-21.

Lai, on the other hand, discloses a method for identifying those objects in an object-oriented programming system that are no longer useful, making the associated memory available for other uses after process using an object has terminated. Lai, Col. 1, line 66-Col. 2, line 2. Apparently, what Lai teaches is an alternative to garbage collection, which Lai deems "expensive". Lai, Col. 1, line 51-65.

The motivation may come from the reference itself or knowledge generally available to one of ordinary skill in the art. MPEP 2143. Considering that Lance and Lai deal with entirely different problems, it is not surprising to find that neither reference provides any hint at what would motivate the person dealing with either the problems solved by Lance or by Lai to make the proposed combination. To combine the teachings of how to remove unnecessary objects (Lai) with the teachings a front-end for program analysis (Lance) makes little sense. The teachings of Lance simply would not benefit from the teachings of Lai or vice versa.

The second source for the motivation is knowledge generally available to one of ordinary skill in the art. "The Examiner must show reasons that the skilled artisan, confronted with the same problems as the inventor and with no knowledge of the claimed invention, would select the elements from the cited prior art references for combination in the manner claim" (*In re Rouffet*, 149 F.3d 1350, 1357 (Fed. Cir.

1998)). "The same problem as the inventor" is, as stated in Appellant's specification, the loading of CAP files onto a smart card while not having the benefit of the size of the method bodies in a Method Component while resolving operands. The problem that arises in that task is that from the method header there is no way to determine the instruction boundary of the associated method body. Appellant has recited this as "determining instruction boundaries of at least one method body" (Claim 1, preamble). That "instruction boundaries" refer to the beginning and end of a method body and not boundaries between basic blocks, is evident from the Specification (*see e.g.*, Specification, Page 6, line 1-3 ("determining the starting point and ending point of the bytecodes containing instructions for each method body") and Fig. 3. and the accompanying text on Page 5, lines 28-30 ("the bytecodes for method bodies 1 335, 2 340, and Y 345, are depicted by 320, 325, and 330, respectively" – it should be noted that 320, 325, and 330 point to the entire bytecode section for each of the method bodies)). Neither reference teaches anything about loading applets onto smart cards. Neither reference teaches anything about resolving operands when loading a Method Component onto a smart card. Thus, it is not surprising that neither reference deals with finding instruction boundaries. Therefore, it is difficult to understand how a person confronted with the problems of the inventor would be motivated to select elements from these references to combine.

The Examiner has made the assertion that "It would have been obvious to one of ordinary skill in the art at the time the invention was made to use Lai's teaching of bounds checking in Lance's algorithm." Office Action, Page 7, lines 3-4. This assertion begs the question "why?" To which the Examiner answers: "One of ordinary skill would have been motivated to track a memory address corresponding to a logical boundary to enhance the analysis of a method, and to avoid inadvertent

programming errors." This assertion fails to address "the reasons that the skilled artisan, confronted with the same problems as the inventor ... would select elements from the cited prior art references" (in re Rouffet). There is no indication in Lance that "logical boundaries" are an issue. In most cases, there is no problem knowing where the boundaries of a program to be analyzed are located. On the contrary, Lance teaches that "A scan is performed for each method in the class, and its Code attribute is obtained" (Lance, 3.2). A code attribute contains a code_length field that "gives the number of bytes in the code array for this method" (The Java Virtual Machine Specification, Chapter 4, The class File Format, Section 4.7.3. The Code Attribute. http://java.sun.com/docs/books/vmspec/2nd-edition/html/ClassFile.doc.html). Thus, having the code_length field available, for Lance, it would not be necessary, and may very well not be desirable to do a bounds check in the manner of Lai because doing so would inherently add additional operations that are not necessary for Lance. Therefore, there would be no motivation to modify Lance to provide the bounds checking of Lai or to otherwise modify Lance to include the limitations of Claim 1.

The third criteria is that the prior art references must teach or suggest all the limitations of the claimed invention. Claim 1 recites three elements:

> a) examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction;
>
> b) maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least

one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected; and

c) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

Consider the second element "maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected." Neither Lance nor Lai teaches or suggests this element. The Examiner has acknowledged that "Lance does not expressly disclose 'b) maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected'" (Office Action, Page 6, lines 14-16).

The Examiner looks to Lai for the second element of Claim 1, citing Lai, Col. 7, Lines 5-11 where it is stated that "The object offset (22) in a virtual address (20) plus the operand size is compared with the size of the reference object on every address translation. This operation is called bounds checking and prevents reference beyond the specified object of a datum which may belong to another object." However, this disclosure in Lai is not equivalent to the claimed element. A "Farthest Logical Return (FLR) Pointer" is claimed as "corresponding to the instruction of the [] method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected" (Claim 1, element b). In other words, while the instructions are being examined (element a), the FLR pointer is being maintained to correspond to either the farthest forward jump or the farthest valid ending instruction.

Lai, on the other hand, teaches that "the size of an object is specified in encoded fields (50) of the object table entry (OTE). The object offset (22) in a virtual address (20) plus the operand size is compared with the size of the referenced object on every address translation." Lai. Col. 7, lines 4-8. Lai is very much in a different domain from the one addressed by Appellant and making any comparison between Appleants' invention and Lai's disclosure is therefore an exercise in comparing the incomparable. That said, consider some of the ways in which what Lai teaches is different from what Appellants claim. First, the bounds of the object in Lai is known *a priori* ("specified in encoded fields") whereas element b of Claim 1 recites a step that allows for determining the instruction boundary while examining instructions in a sequential manner. Second, Lai teaches bounds checking on access of objects ("The object offset (22) in a virtual address (20) plus the operand size is compared with the size of the referenced object on every address translation"). Lai does not teach examining instructions for forward jump and valid ending instructions, which is logical since Lai does not teach anything dealing with instruction analysis. Consequently, Lai does not teach maintaining a pointer that corresponds to farthest forward jump or farthest valid ending instruction. Therefore, a combination of Lance and Lai would not teach or suggest element b of Claim 1.

Claim 1 further recites "c) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer". Lance does not teach or suggest this element. Considering that Lance does not teach or suggest maintaining an FLR pointer, it would be impossible for Lance to terminate examining instructions triggered by a condition dependent on the FLR pointer. Therefore, Lance cannot be considered to teach or suggest element c of Claim 1.

For the foregoing reasons the proffered combination of Lance and Lai has also failed the third requirement for a *prima facie* case of obviousness.

Claims 16, 29, 59, 62, and 77 recite analogous limitations to those set forth in Claim 1. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness for the same reasons given in support of Claim 1.

"If examination at the initial stage does not produce a prima facie case of unpatentability, then without more the applicant is entitled to grant of the patent." In re Oetiker, 977 F.2d 1443, 1445, 24 USPQ2d 1443, 1444 (Fed. Cir. 1992), *quoted in* In re Lowry, 32 F.3d 1579, 32 USPQ2d 1031 (Fed. Cir. 1994). Thus, for the reasons given above, Appellants respectfully request reversal of the rejection of Claims 1,16, 29, 59, 62, and 77 and their early allowance.

Claims 2, 5, 8, 11, 20, 23, 26, 46, 49, 53, 56, 64, 67, 71, and 74

This group of claims stand rejected under 35 USC 103(a) as unpatentable over Lance in view of Lai. Appellants respectfully request the reversal of the rejection of this group of Claims.

Claim 2 is representative of this group of claims as the other claims in the group contain limitations analogous to those recited in Claim 2.

The claims in this group incorporate all the limitations of their respective base claims which are argued patentable over Lance and Lai herein above, and are patentable for the reasons given in support of their base claims.

The claims in this group provide further unique and non-obvious combinations and are patentable over Lance and Lai by virtue of such further combinations.

Claim 2 recites "setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer." As discussed hereinabove, the Examiner has failed to establish a *prima facie* case of obviousness with respect to Claim 1 for the reason that there is no motivation to combine the Lance and Lai references. That argument also applies to Claim 2.

Furthermore, the proffered combination of Lance and Lai would also fail to provide a motivation for modifying Lance to "setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer" (Claim 2). As noted in the argument in support of Claim 1, the combination of Lance and Lai fail to teach or suggest maintaining an FLR Pointer. Accordingly, it logically follows that Lance and Lai fail to teach or suggest this recited limitation. Furthermore, Lance teaches the analysis of a Java Class file. The class file structure includes all information necessary to determine method boundaries. The Java Virtual Machine Specification, Chapter 4, The class File Format, Section 4.1. The ClassFile Structure. http://java.sun.com/docs/books/vmspec/2nd-edition/html/ClassFile.doc.html. Therefore, there would be no need in Lance to modify the disclosure therein to include "setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer".

A criteria for establishment of a *prima facie* case for obviousness is that the prior art references must teach or suggest all the limitations of the claimed invention. Claim 2 recites "setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer". Neither Lance nor Lai teaches or suggests this limitation. The Examiner has pointed to Section 3.2 of Lance where it is stated that "A scan is performed for each method in

the class, and its Code attribute is obtained". However, there is no indication in that passage that either a start pointer or an FLR Pointer is maintained. The Examiner suggests that Lai provides *motivation* for using pointers to track memory addresses. However, while Lai provides a disclosure permitting bounds checking of objects, that cannot be extended to setting a start pointer to a next method body. In an object oriented programming system, there is no way of knowing where in memory particular objects are located. That should be particularly evident from a study of Lai considering that that reference deals with memory reclamation from objects that are no longer in use. Therefore, if an out-of-bounds has been detected in Lai, that only gives an indication of an error condition, not where a next object in a sequence of objects may be located. Thus, anything learned from Lai would not yield an extension to Lance of "setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer".

For this reason, Claim 2 is not obvious over the combination of Lance and Lai, taken singly or in combination.

Claims 5, 8, 11, 20, 23, 26, 46, 49, 53, 56, 64, 67, 71, and 74 recite analogous limitations to those set forth in Claim 2. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness for the same reasons given in support of Claim 1. Thus, for the reasons given above, Appellants respectfully request reversal of the rejection of Claims 2, 5, 8, 11, 20, 23, 26, 46, 49, 53, 56, 64, 67, 71, and 74 and their early allowance.

Claims 3, 6, 9, 12, 18, 21, 24, 27, 34, 40, 50, 54, 57, 65, 68, 72, 75

This group of claims stand rejected under 35 USC 103(a) as unpatentable over Lance in view of Lai. Appellants respectfully request the reversal of the rejection of this group of Claims.

Claim 3 is representative of this group of claims as the other claims in the group contain limitations analogous to those recited in Claim 3.

The claims in this group incorporate all the limitations of their respective base claims, which are argued patentable over Lance and Lai herein above, and are patentable for the reasons given in support of their base claims.

The claims in this group provide further unique and non-obvious combinations and are patentable over Lance and Lai by virtue of such further combinations.

Claim 3 recites "the steps [recited in the base and intervening claims] are performed on each of a successive method bodies of the at least one method body." As discussed hereinabove, the Examiner has failed to establish a *prima facie* case of obviousness with respect to Claim 1 for the reason that there is no motivation to combine the Lance and Lai references. That argument also applies to Claim 3.

While Lance teaches that "a scan is performed for each method in the class", considering that Lance does not teach or suggest the elements of Claims 1 and 2 (and as will be seen of the other intervening claims), inherently Lance and Lai do not teach or suggest performing those elements on successive method bodies.

For this reason, Claim 2 is not obvious over the combination of Lance and Lai, taken singly or in combination.

Claims 6, 9, 12, 18, 21, 24, 27, 34, 40, 50, 54, 57, 65, 68, 72, 75 recite analogous limitations to those set forth in Claim 3. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness for the same reasons given in

support of Claim 3. Thus, for the reasons given above, Appellants respectfully request reversal of the rejection of Claims 3, 6, 9, 12, 18, 21, 24, 27, 34, 40, 50, 54, 57, 65, 68, 72, 75 and their early allowance.

Claims 4, 19, 32, 48, and 66

The claims in this group provide further unique and non-obvious combinations and are patentable over Lance, Lai and Baentsch by virtue of such further combinations.

Claim 4 recites "d) resolving each unresolved reference in each instruction of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer." In light of the arguments presented herein above, it must be clear that Lance and Lai do not teach "resolving ... starting with the first instruction of [the] method body and ending with the instruction corresponding to the FLR Pointer" because Lance and Lai fail to teach or suggest maintaining an FLR Pointer.

Baentsch teaches an system of loading and linking a CAP file that has a modified constant pool with information of use during linking. Baentsch, Abstract. Baentsch does not teach or suggest maintaining an FLR pointer as set forth in Claim 1. Baentsch's system provides a "fixup table again contains the position in the text or data section where a relocation has to take place" (Baentsch, Page 4, lines 12-13). Thus, Baensch knows where the relocations are to be performed and would therefore not need to keep track of instruction boundaries as proposed by Applicants. Therefore, it is not surprising that Baentsch also fails to teach or suggest "resolving ...

starting with the first instruction of [the] method body and ending with the instruction corresponding to the FLR Pointer."

There is no motivation to combine Baentsch with Lance. Lance teaches a program-analysis method. Program analysis can operate quite well without knowing anything about how to resolve operands. In fact, it could be desirable to perform program-analysis on programs prior to performing operand resolution. Therefore, Lance would not be enhanced in any way by adding the teaching of Baentsch thereto. Conversely, in most cases, operand resolution works very well without doing any form of program analysis. Thus, there would be no motivation to modify Baentsch to perform the operations described in Lance.

For this reason, Claim 4 is not obvious over the combination of Lance and Lai, taken singly or in combination.

Claims 19, 32, 48, and 66 recite analogous limitations to those set forth in Claim 2. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness with respect to these claims for the same reasons given in support of Claim 2. Thus, for the reasons given above, Appellants respectfully request reversal of the rejection of Claims 4, 19, 32, 48, and 66 and their early allowance.

Claims 7, 10, 22, 25, 35, 38, 51, 53, 55, 69, and 73

This group of claims stand rejected under 35 USC 103(a) as unpatentable over Lance in view of Lai and further in view of JCVMS 2.1, Lance in view of Lai and further in view of JCVMS 2.1 in view of Baentsch. Appellants respectfully request the reversal of the rejection of this group of Claims.

Claim 7 is representative of this group of claims as the other claims in the group contain limitations analogous to those recited in Claim 7 or depend from a claim that includes limitations analogous to those recited in Claim 7.

The claims in this group incorporate all the limitations of their respective base claims, which are argued patentable over Lance and Lai herein above, and are patentable for the reasons given in support of their base claims. JVMCS 2.1 and Baentsch do not provide a teaching or suggestion of the limitations lacking from the combination of Lance and Lai nor provide a motivation to modify or combine those references in a manner to negate the argument provided in support of the base claims. As noted in the Applicants' Specification, the Java Card Specification provides for a Reference Relocation Component. JCVMS 2.1, 6.11. Generally speaking, the JCVMS does not provide implementation details. However, considering that the specification provides description of the Reference Relocation Component, which is what Applicants manage to avoid using by employing the claimed method for determining instruction boundaries by maintaining an FLR pointer and terminating examination of instruction in response to examining an instruction beyond the FLR pointer, the inference to be drawn is that JCVMS 2.1 would suggest to the skilled artisan to use the Reference Relocation Component for linking purposes. Therefore, there would be no suggestion drawn from JCVMS 2.1 to perform the steps claimed in Claim 1.

Accordingly, Claim 7 is patentable over Lance, Lai, JCVMS 2.1 and Baentsch for the reasons given in support of Claim 1.

Furthermore, Claim 7 recites "examining in a sequential manner each instruction of the exception handler code corresponding to the at least one method body starting with a first instruction of the exception handler code corresponding to

the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction; (ii) maintaining a FLR Pointer corresponding to the exception handler code corresponding to the at least one method body for which the farthest forward jump or the farthest valid ending instruction is detected; and (iii) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer." As discussed hereinabove, the Examiner has failed to establish a *prima facie* case of obviousness with respect to Claim 1 for the reason that there is no motivation to combine the Lance and Lai references. That argument also applies to Claim 7.

Elements *i*, *ii*, and *iii* of Claim 7 are analogous to elements a, b, and c of Claim 1 other than that it is to an exception handler code that these claims perform the steps of examining for forward jumps and valid ending instruction, maintaining an FLR Pointer corresponding thereto, and terminating examining for a forward jump instruction or ending instruction when the instruction under examination is beyond the FLR Pointer. As discussed herein above in support of Claim 1, Lance and Lai fail to teach or suggest these elements as applied generally. Lance and Lai also fail to teach or suggest applying these steps to an exception handler.

Baentsch and JCVMS 2.1 also fail to teach or suggest "maintaining a FLR Pointer corresponding to the exception handler code corresponding to the at least one method body for which the farthest forward jump or the farthest valid ending instruction is detected" and "terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer" (Claim 7). Therefore, the combination of Lance,

Lai, Baentsch, and JSVMS 2.1 in any permutation would fail to teach or suggest the claimed invention.

The Examiner argues that JSVMS 2.1 requires proper handling of exceptions and that there would be a motivation to examine all code in a program in order to implement exceptions according to the specification (Office Action, Page 10). However, it does not follow from that assertion that examination of the exception handler would require maintaining an FLR Pointer and terminating examination of the exception handler when the next instruction under examination is beyond the FLR pointer.

For this reason, Claim 2 is not obvious over the combination of Lance and Lai, taken singly or in combination.

Claims 10, 22, 25, 35, 38, 51, 53, 55, 69, and 73 recite analogous limitations to those set forth in Claim 7. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness with respect to these claims for the same reasons given in support of Claim 7. Thus, for the reasons given above, Appellants respectfully request reversal of the rejection of Claims 7, 10, 22, 25, 35, 38, 51, 53, 55, 69, and 73 and their early allowance.

The Remaining Claims

All remaining claims not argued explicitly hereinabove depend from one or more of the claims argued hereinabove and incorporate the limitations of such independent or intervening claims. None of the references cited against these dependent claims, i.e., JCVMS 2.1.1, Lindholm, and Chen, provide the motivation to combine Lance, Lai, Baentsch, or JCVMS 2.1, in any suggested permutation, that is

not found in Lance, Lai, Baentsch, or JCVMS 2.1. Furthermore, the Examiner has not asserted that the motivation to combine Lance, Lai, Baentsch and JCVMS 2.1, in any permutation, may be found in JCVMS 2.1, Lindholm, and Chen. Therefore, the failure of the *prima facie* case against the claims argued herein above stands with respect to these further references.

These further references also fail to teach or suggest the elements from the claims argued hereinabove. JCVMS 2.1.1 and Chen are similar in scope to JCVMS 2.1. Therefore the argument presented herein above with respect to JCVMS 2.1 applies to these references as well. Lindholm is the "Java Virtual Machine Specification". As described herein above with respect to Lance, the Java Virtual Machine Specification contains a code_attribute that specifies the length for the associated method. Thus, Lindholm, like Lance, would not lead to the necessity for determining instruction boundaries in the manner claimed in Claim 1. Accordingly, the claims argued hereinabove are patentable over any combination of the cited references for the reasons given. Furthermore, the claims not argued explicitly hereinabove recite further unique and non-obvious combinations and are patentable both for the same reasons given in support of their respective base claims and any intervening claims and by virtue of such further combinations.

Thus, for the reasons given above, Appellants respectfully request reversal of the rejection of claims not argued explicitly hereinabove and their early allowance.

Conclusion of Argument

A proper rejection under 35 USC 103(a) requires that the Examiner establishes an unrebutted *prima facie* case of obviousness. As discussed hereinabove, the

Examiner has filed to establish a *prima facie* case of obviousness. "If examination at the initial stage does not produce a prima facie case of unpatentability, then without more the applicant is entitled to grant of the patent." In re Oetiker, 977 F.2d 1443, 1445, 24 USPQ2d 1443, 1444 (Fed. Cir. 1992), *quoted in* In re Lowry, 32 F.3d 1579, 32 USPQ2d 1031 (Fed. Cir. 1994). Accordingly, Appellants respectfully request reversal of the rejection of the Claims and their early allowance.

Respectfully Submitted,

Pehr Jansson
Reg. No. 35,759

Date: Nov 2, 2005

Pehr Jansson
Anderson & Jansson, L.L.P.
9501 N. Capital of TX Hwy. #202
Austin, TX 78759

512-372-8440 x200
678-868-0101 (fax)
pehr@anjanlaw.com

8. Appendix 1: Copy of Claims on Appeal

1. A method for determining instruction boundaries of at least one method body within a computer code loaded into a memory of a smart card comprising:

a) examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction;

b) maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected; and

c) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

2. The method of claim 1 further comprising:

d) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

3. The method of claim 2 wherein each of the steps a), b), c), and d) are performed on each of a successive method bodies of the at least one method body.

4. The method of claim 1 further comprising:

d) resolving each unresolved reference in each instruction of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer.

5. The method of claim 4 further comprising:

e) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

6. The method of claim 5 wherein each of the steps a), b), c), d), and e) are performed on each of a successive method bodies of the at least one method body.

7. The method of claim 1 further comprising:

d) determining if an exception handler code corresponding to the at least one method body is present; and

e) responsive to determining that an exception handler code corresponding to the at least one method body is present beyond the instruction corresponding to the FLR Pointer:

(i) examining in a sequential manner each instruction of the exception handler code corresponding to the at least one method body starting with a first instruction of the exception handler code corresponding to the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction;

(ii) maintaining a FLR Pointer corresponding to the exception handler code corresponding to the at least one method body for which the farthest forward jump or the farthest valid ending instruction is detected; and

(iii) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

8. The method of claim 7 further comprising:

f) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

9. The method of claim 8 wherein each of the steps a), b), c), d), e), and f) are performed on each of a successive method bodies of the at least one method body.

10. The method of claim 7 further comprising:

f) resolving each unresolved reference in each instruction of the exception handler code corresponding to the at least one method body starting with the first instruction of the exception handler code corresponding to the at least one method body and ending with the instruction corresponding to the FLR Pointer.

11. The method of claim 10 further comprising:

g) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

12. The method of claim 11 wherein each of the steps a), b), c), d), e), f), and g) are performed on each of a successive method bodies of the at least one method body.

13. The method of claim 7 wherein determining if an exception handler code corresponding to the at least one method body is present includes:

(i) determining if an exception entry is available in an exception handler array corresponding to the computer code;

(ii) responsive to determining that an exception entry is available, determining a catch range for the exception entry; and

(iii) determining if any of the instructions of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer are within the catch range.

14. The method of claim 1 wherein the computer code comprises a methods item of a method component of a converted applet file.

15. The method of claim 1 wherein the memory comprises non-volatile read/write memory.

16. A computer-readable medium tangibly having a program of machine-readable instructions for causing a processor to perform a method for determining instruction boundaries of at least one method body within a computer code loaded into a memory of a smart card, the method comprising:

a) examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction;

b) maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected; and

c) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

17. The computer-readable medium of claim 16 further having instructions for causing a processor to perform a method, the method comprising:

d) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

18. The computer-readable medium of claim 17 further having instructions for causing a processor to perform each of the steps a), b), c), and d) on each of a successive method bodies of the at least one method body.

19. The computer-readable medium of claim 16 further having instructions for causing a processor to perform a method, the method comprising:

d) resolving each unresolved reference in each instruction of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer.

20. The computer-readable medium of claim 19 further having instructions for causing a processor to perform a method, the method comprising:

e) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

21. The computer-readable medium of claim 20 further having instructions for causing a processor to perform each of the steps a), b), c), d), and e) on each of a successive method bodies of the at least one method body.

22. The computer-readable medium of claim 16 further having instructions for causing a processor to perform a method, the method comprising:

d) determining if an exception handler code corresponding to the at least one method body is present; and

e) responsive to determining that an exception handler code corresponding to the at least one method body is present beyond the instruction corresponding to the FLR Pointer:

(i) examining in a sequential manner each instruction of the exception handler code corresponding to the at least one method body starting with a first instruction of the exception handler code corresponding to the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction;

(ii) maintaining a FLR Pointer corresponding to the exception handler code corresponding to the at least one method body for which the farthest forward jump or the farthest valid ending instruction is detected; and

(iii) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

23. The computer-readable medium of claim 22 further having instructions for causing a processor to perform a method, the method comprising:

f) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

24. The computer-readable medium of claim 23 further having instructions for causing a processor to perform each of the steps a), b), c), d), e), and f) on each of a successive method bodies of the at least one method body.

25. The computer-readable medium of claim 22 further having instructions for causing a processor to perform a method, the method comprising:

f) resolving each unresolved reference in each instruction of the exception handler code corresponding to the at least one method body starting with the first instruction of the exception handler code corresponding to the at least one method body and ending with the instruction corresponding to the FLR Pointer.

26. The computer-readable medium of claim 25 further having instructions for causing a processor to perform a method, the method comprising:

g) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

27. The computer-readable medium of claim 26 further having instructions for causing a processor to perform each of the steps a), b), c), d), e), f), and g) on each of a successive method bodies of the at least one method body.

28. The computer-readable medium of claim 22 wherein the instructions for determining if an exception handler code corresponding to the at least one method body is present includes instructions for causing a processor to perform a method, the method comprising:

(i) determining if an exception entry is available in an exception handler array corresponding to the computer code;

(ii) responsive to determining that an exception entry is available, determining a catch range for the exception entry; and

(iii) determining if any of the instructions of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer are within the catch range.

29. A smart card configured to receive computer code having at least one method body within the computer code comprising:

a memory;

a processor connected to the memory; and

an installer module having logic operable to cause the processor to receive the computer code into the memory; and further having logic operable to cause the processor to determine instruction boundaries within the computer code by a) examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction; b) maintaining a Farthest Logical Return (FLR) Pointer corresponding to the instruction of the at least one method body for which the farthest forward jump instruction or the farthest valid ending instruction is detected; and c) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

30. The smart card of claim 29 further having logic operable to cause the processor to determine instruction boundaries within the computer code by d) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

31. The smart card of claim 30 wherein each of the steps a), b), c), and d) are performed on each of a successive method bodies of the at least one method body.

32. The smart card of claim 29 further having logic operable to cause the processor to resolve unresolved references in the at least one method body by d) resolving each unresolved reference in each instruction of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer.

33. The smart card of claim 32 further having logic operable to cause the processor to determine instruction boundaries within the computer code by e) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

34. The smart card of claim 33 wherein each of the steps a), b), c), d), and e) are performed on each of a successive method bodies of the at least one method body.

35. The smart card of claim 29 further having logic operable to cause the processor to determine instruction boundaries within the computer code by d) determining if an exception handler code corresponding to the at least one method body is present; and e) responsive to determining that an exception handler code corresponding to the at least one method body is present beyond the instruction corresponding to the FLR Pointer: (i) examining in a sequential manner each instruction of the exception handler code corresponding to the at least one method body starting with a first instruction of the exception handler code corresponding to the at least one method body for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction; (ii) maintaining a FLR Pointer corresponding to the exception handler code corresponding to the at least one method body for which the farthest forward jump or the farthest valid ending instruction is detected; and (iii) terminating the examining for a forward jump or a valid ending instruction when the instruction under examination is beyond the instruction corresponding to the FLR Pointer.

36. The smart card of claim 35 further having logic operable to cause the processor to determine instruction boundaries within the computer code by f) setting a Start

Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

37. The smart card of claim 36 wherein each of the steps a), b), c), d), e), and f) are performed on each of a successive method bodies of the at least one method body.

38. The smart card of claim 35 further having logic operable to resolve unresolved reference in the exception handler code corresponding to the at least one method body by f) resolving each unresolved reference in each instruction of the exception handler code corresponding to the at least one method body starting with the first instruction of the exception handler code corresponding to the at least one method body and ending with the instruction corresponding to the FLR Pointer.

39. The smart card of claim 38 further having logic operable to cause the processor to determine instruction boundaries within the computer code by g) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

40. The smart card of claim 39 wherein each of the steps a), b), c), d), e), f), and g) are performed on each of a successive method bodies of the at least one method body.

41. The smart card of claim 35 further having logic operable to cause the processor to determine if an exception handler code corresponding to the at least one method body is present by (i) determining if an exception entry is available in an exception handler array corresponding to the computer code; (ii) responsive to determining that an exception entry is available, determining a catch range for the exception entry; and (iii) determining if any of the instructions of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer are within the catch range.

42. The smart card of claim 29 wherein the computer code comprises a methods items of a method component of a converted applet file.

43. The smart card of claim 29 wherein the memory comprises non-volatile read/write memory.

44. A method for determining instruction boundaries of at least one method body within a computer code loaded into a memory of a smart card comprising:

a) examining the instructions of the at least one method body to determine a farthest logical return within the at least one method body;

b) establishing the instruction boundary at the instruction located at the farthest logical return; and

c) terminating the examination of the instructions when the instruction under examination is beyond the farthest logical return.

45. The method of claim 44 wherein the examining step (a) includes examining for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction.

46. The method of claim 44 further comprising:

d) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the farthest logical return.

47. The method of claim 46 wherein each of the steps a), b), c), and d) are performed on each of a successive method bodies of the at least one method body.

48. The method of claim 44 further comprising:

d) resolving each unresolved reference in each instruction of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the farthest logical return.

49. The method of claim 48 further comprising:

e) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the farthest logical return.

50. The method of claim 49 wherein each of the steps a), b), c), d), and e) are performed on each of a successive method bodies of the at least one method body.

51. The method of claim 44 further comprising:

d) determining if an exception handler code corresponding to the at least one method body is present; and

e) responsive to determining that an exception handler code corresponding to the at least one method body is present beyond the instruction corresponding to the farthest logical return:

(i) examining the instructions of the exception handler code corresponding to the at least one method body to determine a farthest logical return within the exception handler code corresponding to the at least one method body;

(ii) establishing the instruction boundary at the instruction located at the farthest logical return within the exception handler code corresponding to the at least one method body; and

(iii) terminating the examination of the instructions when the instruction under examination is beyond the farthest logical return within the exception handler code corresponding to the at least one method body.

52. The method of claim 51 wherein the examining step (e)(i) includes examining for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction.

53. The method of claim 51 further comprising:

f) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the farthest logical

return within the exception handler code corresponding to the at least one method body.

54. The method of claim 53 wherein each of the steps a), b), c), d), e), and f) are performed on each of a successive method bodies of the at least one method body.

55. The method of claim 51 further comprising:

f) resolving each unresolved reference in each instruction of the exception handler code corresponding to the at least one method body starting with the first instruction of the exception handler code corresponding to the at least one method body and ending with the instruction corresponding to the farthest logical return within the exception handler code corresponding to the at least one method body.

56. The method of claim 55 further comprising:

g) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the farthest logical return within the exception handler code corresponding to the at least one method body.

57. The method of claim 56 wherein each of the steps a), b), c), d), e), f), and g) are performed on each of a successive method bodies of the at least one method body.

58. The method of claim 51 wherein determining if an exception handler code corresponding to the at least one method body is present includes:

(i) determining if an exception entry is available in an exception handler array corresponding to the computer code;

(ii) responsive to determining that an exception entry is available, determining a catch range for the exception entry; and

(iii) determining if any of the instructions of the at least one method body starting with the first instruction of the at least

one method body and ending with the instruction corresponding to the farthest logical return are within the catch range.

59. A smart card configured to receive computer code having at least one method body within the computer code comprising:

a memory;

a processor connected to the memory; and

an installer module having logic operable to cause the processor to receive the computer code into the memory; and further having logic operable to cause the processor to determine instruction boundaries within the computer code by a) examining the instructions of the at least one method body to determine a farthest logical return within the at least one method body; b) establishing the instruction boundary at the instruction located at the farthest logical return; and c) terminating the examination of the instructions when the instruction under examination is beyond the farthest logical return.

60. The smart card of claim 59 wherein the computer code comprises a methods item of a method component of a converted applet file.

61. The method of claim 44 wherein the memory comprises non-volatile read/write memory.

62. A method for determining instruction boundaries of at least one method body within a computer code loaded into a memory of a smart card comprising:

(a) examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction that establishes a logical return;

(b) maintaining a Farthest Logical Return (FLR) Pointer to continuously store the farthest logical return found in the examining step (a); and

(c) terminating examination of the instructions when an instruction under examination is beyond the instruction corresponding to the FLR Pointer.

63. The method of claim 62 wherein the examining for an instruction that establishes a logical return of step (a) includes examining for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction.

64. The method of claim 63 further comprising:

d) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

65. The method of claim 64 wherein each of the steps a), b), c), and d) are performed on each of a successive method bodies of the at least one method body.

66. The method of claim 62 further comprising:

d) resolving each unresolved reference in each instruction of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer.

67. The method of claim 66 further comprising:

e) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

68. The method of claim 67 wherein each of the steps a), b), c), d), and e) are performed on each of a successive method bodies of the at least one method body.

69. The method of claim 62 further comprising:

d) determining if an exception handler code corresponding to the at least one method body is present; and

e) responsive to determining that an exception handler code corresponding to the at least one method body is present beyond the instruction corresponding to the FLR Pointer:

(i) examining in a sequential manner each instruction of the exception handler code corresponding to the at least one method body starting with a first instruction of the exception handler code corresponding to the at least one method body for an instruction that establishes a logical return;

(ii) maintaining a FLR Pointer to continuously store the farthest logical return found in the examining step e(i);

(iii) terminating examination of the instructions when an instruction under examination is beyond the instruction corresponding to the FLR Pointer.

70. The method of claim 69 wherein the examining step (e)(i) includes examining for an instruction selected from a group consisting of a forward jump instruction and a valid ending instruction.

71. The method of claim 69 further comprising:

f) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

72. The method of claim 71 wherein each of the steps a), b), c), d), e), and f) are performed on each of a successive method bodies of the at least one method body.

73. The method of claim 69 further comprising:

f) resolving each unresolved reference in each instruction of the exception handler code corresponding to the at least one method body starting with the first instruction of the exception handler code corresponding to the at least one method body and ending with the instruction corresponding to the FLR Pointer.

74. The method of claim 73 further comprising:

g) setting a Start Pointer to a next method body of the at least one method body following the instruction corresponding to the FLR Pointer.

75. The method of claim 74 wherein each of the steps a), b), c), d), e), f), and g) are performed on each of a successive method bodies of the at least one method body.

76. The method of claim 69 wherein determining if an exception handler code corresponding to the at least one method body is present includes:

(i) determining if an exception entry is available in an exception handler array corresponding to the computer code;

(ii) responsive to determining that an exception entry is available, determining a catch range for the exception entry; and

(iii) determining if any of the instructions of the at least one method body starting with the first instruction of the at least one method body and ending with the instruction corresponding to the FLR Pointer are within the catch range.

77. A smart card configured to receive computer code having at least one method body within the computer code comprising:

a memory;

a processor connected to the memory; and

an installer module having logic operable to cause the processor to receive the computer code into the memory; and further having logic operable to cause the processor to determine instruction boundaries within the computer code by a) examining in a sequential manner each instruction of the at least one method body starting with a first instruction of the at least one method body for an instruction that establishes a logical return; b) maintaining a Farthest Logical Return (FLR) Pointer to continuously store the farthest logical return found in the examining step (a); and c) terminating examination of the

instructions when an instruction under examination is beyond the instruction corresponding to the FLR Pointer.

78. The smart card of claim 77 wherein the computer code comprises a methods item of a method component of a converted applet file.

79. The method of claim 77 wherein the memory comprises non-volatile read/write memory.